Speeded Up Robust Features (SURF): Performance test

Manuel Benito Sayago 764666

Speeded Up Robust Features (SURF)

INDEX

NOTE: Working with MATLAB 7.10 (r2010a), Visual studio 2010, SURF 1.0.9 and OpenCV 2.1.0

Abstract

This work presents a performance analysis of SURF features, an algorithm for feature detection and matching. The goal is to test the performance of SURF in the presence of noise: The analysis is performed both on synthetically generated observations as well on raw images. In first place we present SURF features [1]. Introduction covers the concept of feature extraction, what it is and the interest of it, as well the feature points detection, description and matching.

After the introduction we proceed with the main part: Experiments. In the first part we use test images and we add noise (additive noise). The additive noise we work with is Gaussian and Poisson noise. We do this process several times and in each iteration we compute the SURF features of the original image and the altered one. Afterwards the performance is analyzed, considering the repeatability and the ratio of incorrect matches. We also consider the changes of illumination, out of focus blur and motion blur. Finally conclusions are taken from the results.

In the second part the experiments are done with raw files. A raw file is the data obtained from the sensor, without changes, so it needs a bit of processing for be able even to preview it. It is analyzed the performance of SURF when it is changed the exposure time. In this case noise will be as a result of the camera defects and noise related to the exposure time. After the experiments, the results are analyzed as in the first part (repeatability and ratio of incorrect matches).

It results that SURF is robust to noise in a wide range, and out of this range the results are poor and inaccurate. It is sensible to blur and the performance is bad in dark environments (pictures).

An appendix is written for explain the process of port the SURF libraries, which are originally written in C and ready for use with OpenCV in any platform that admits OpenCV (such as C, python or Java) and make them usable in Matlab. For this purpose Matlab has a tool that is called Mex file. This tool is explained and the process in general.

SURF FEATURES

1. INTRODUCTION

In image processing sometimes there is an interest to detect points or regions in an image. For this purpose there are algorithms to detect and describe local features in images. For any object in an image, interesting points on the object can be extracted to provide a "feature description" of the object. This description, extracted from the image, can be used to identify the object when attempting to locate the object in a test image containing many other objects. Task of finding correspondences has many applications: Image registration, object recognition, image retrieval, 3D reconstruction, for mention some examples.

These features are points. Interest points are usually distinctive locations in the image such as corners, blobs and T-junctions. The most valuable factor of an interest point detector is repeatability. Repeatability expresses the reliability of finding the same physical interest points under different viewing conditions. The points are defined by a descriptor, which is a vector that contains information about the point itself and the surroundings. Not only the local gradients but also the direction and the sign are contained in this feature vector. This descriptor has to be robust to noise and at the same time distinctive. Finally the descriptor vectors are matched between different images. The matching is based on a distance between the vectors, e.g. Euclidean distance. Not only is the distance considered but also the sign of the Laplacian. The search of interest points can be divided in three main steps: Detect interest points, the neighborhoods of the points are represented with a descriptor and finally the vectors are matched. These steps will be explained in detail later.

Another important characteristic of these features is that the relative positions between them in the original scene shouldn't change from one image to another. For example, if only the four corners of a door were used as features, they would work regardless of the door's position; but if points in the frame were also used, the recognition would fail if the door is opened or closed. Similarly, features located in articulated or flexible objects would typically not work if any change in their internal geometry happens between two images in the set being processed. However, in practice SURF detects and uses a much larger number of features from the images, which reduces the contribution of the errors caused by these local variations in the average error of all feature matching errors. SURF can robustly identify objects even among clutter and under partial occlusion, because his SURF feature descriptor is invariant to scale, orientation, and affine distortion, and partially invariant to illumination changes.

2. RELATED WORK.

2.1. Interest point detection.

The most widely used detector probably is the Harris corner detector [6]. It is based on the eigenvalues of the second moment matrix. However, this detector is not scale invariant. Lindeberg introduced the concept of automatic scale selection [7]. Also he experimented with both the determinant of the Hessian matrix as well as the Laplacian. Mikolajczyk and Schmid [9] created a robust scale-invariant feature detector, which they coined Harris-Laplace and Hessian-Laplace.

Studying the existing detectors and from all the published comparisons [10, 11] we can conclude that the Hessian-based detectors are more stable and repeatable than the Harris-based counterparts. Moreover, using the determinant rather than its trace (the Laplacian) seems advantageous.

2.2. Interest point description.

A large variety of feature descriptors have been proposed. However, the descriptor introduced by Lowe [24] has been shown to outperform the others. This can be explained by the fact that they capture a substantial amount of information about the intensity patterns, while at the same time being robust to small deformations. The descriptor, called SIFT (Scale-Invariant Feature Transform) for short, computes a histogram of local gradients around the interest point and stores the bins in a 128 dimensional vector (8 orientation bins for each of 4x4 location bins).

3. INTEREST POINT DETECTION

An interest point is a featured location in an image. It is usually in a corner or in a T junction. For the detection it is used the Hessian-matrix approximation and integral images.

3.1. Integral images.

They allow for fast computation of box type convolution filters**.** In an integral image the value of a pixel (x, y) is the sum of all the pixels in the square region within the origin and the pixel (x, y). It takes only three additions and four memory access to calculate the sum of intensities inside a rectangular region of any size.

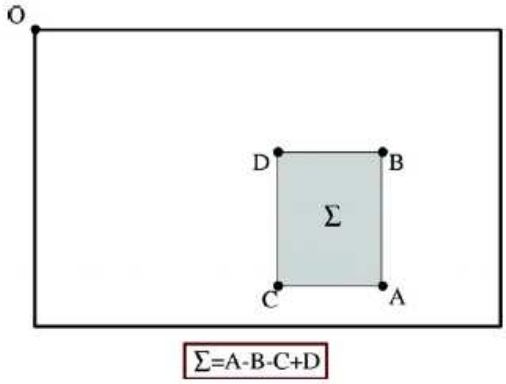


Fig. 1. Using integral images, it takes only three additions and four memory accesses to calculate the sum of intensities inside a rectangular region of any size.

3.2. Hessian matrix based interest points.

The detector is based on the Hessian matrix. We detect blob-like structures at locations were the determinant is maximum. In contrast to the Hessian-Laplace detector by Mikolajczyk and Schmid [9]. We rely on the determinant of the Hessian also for the scale selection.

$$\mathcal{H}(\mathbf{x},\sigma) = \begin{bmatrix} L_{xx}(\mathbf{x},\sigma) & L_{xy}(\mathbf{x},\sigma) \\ L_{xy}(\mathbf{x},\sigma) & L_{yy}(\mathbf{x},\sigma) \end{bmatrix}$$

Hessian matrix, where Lxx(x, σ) is the convolution of the Gaussian second order derivative at point x and scale sigma, and Dxx is the discretised version.

Gaussians are optimal for scale-space analysis, but in practice they have to be discretised and cropped, since we are working with digital values. This leads to a loss in repeatability under image rotations around odd multiples of pi/4. This weakness holds for Hessian-based detectors in general. This is due to the square shape of the filter. Nevertheless, the detectors still perform well, and the slight decrease in performance does not outweigh the advantage of fast convolutions brought by the discretisation and cropping.
These approximate second order Gaussian derivatives and can be evaluated at a very low computational cost using integral images. The calculation time therefore is independent of the filter size.

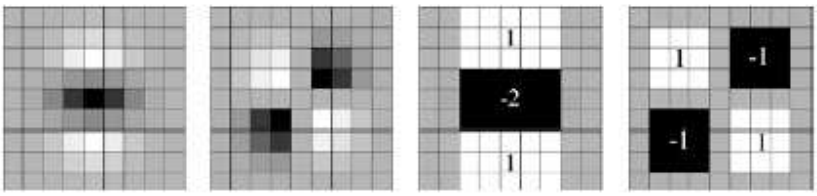


Fig. 2. Left to right: The (discretised and cropped) Gaussian second order partial derivative in y- (L_{yy}) and xy-direction (L_{xy}), respectively; our approximation for the second order Gaussian partial derivative in y- (D_{yy}) and xy-direction (D_{xy}). The grey regions are equal to zero.

$$\det(H_{approx}) = \mathrm{DxxDyy} - (\mathrm{wDxy})^2$$

Where Dxx is the discretised Gaussian second order derivative

The relative weight w of the filter responses is used to balance the expression for the Hessian's determinant. This is needed for the energy conservation between the Gaussian kernels and the approximated Gaussian kernels, where |x| is the Frobenius norm. Notice that for theoretical correctness, the weighting changes depending on the scale. In practice, we keep this factor constant, as this did not have a significant impact on the results in our experiments.

Furthermore, the filter responses are normalised with respect to their size. This guarantees a constant Frobenius norm for any filter size, an important aspect for the scale space analysis as discussed in the next section. The approximated determinant of the Hessian represents the blob response in the image at location x. These responses are stored in a blob response map over different scales, and local maxima are detected as explained in Section 3.4.

3.3. Scale space representation.

Interest points need to be found at different scales, not least because the search of correspondences often requires their comparison in images where they are seen at different scales. Scale spaces are usually implemented as an image pyramid. The images are repeatedly smoothed with a Gaussian and then sub-sampled in order to achieve a higher level of the pyramid. Lowe subtracts these pyramid layers in order to get the DoG (Difference of Gaussians) images where edges and blobs can be found.

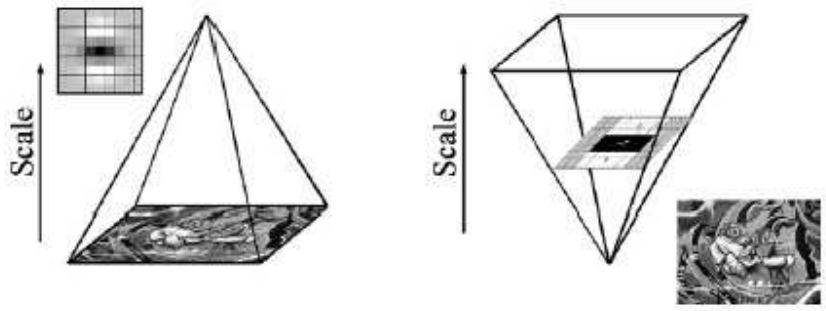


Fig. 4. Instead of iteratively reducing the image size (left), the use of integral images allows the up-scaling of the filter at constant cost (right).

Due to the use of box filters and integral images, we do not have to iteratively apply the same filter to the output of a previously filtered layer, but instead can apply box filters of any size at exactly the same speed directly on the original image and even in parallel (although the latter is not exploited here). Therefore, the scale space is analyzed by up-scaling the filter size rather than iteratively reducing the image size

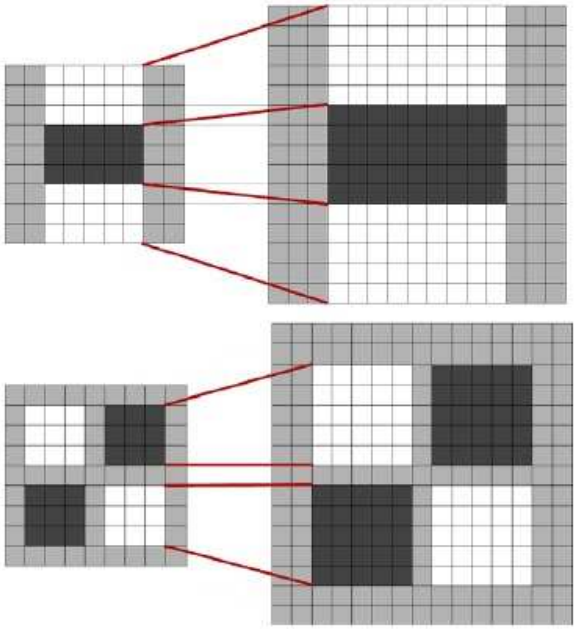


Fig. 5. Filters D_{yy} (top) and D_{xy} (bottom) for two successive scale levels (9 × 9 and 15 × 15). The length of the dark lobe can only be increased by an even number of pixels in order to guarantee the presence of a central pixel (top).

Therefore, the scale space is analyzed by up-scaling the filter size rather than iteratively reducing the image size. The output of the 9 x 9 filter, introduced in previous section, is considered as the initial scale layer, to which we will refer as scale s=1.2 (approximating Gaussian derivatives with sigma=1.2). The following layers are obtained by filtering the image with gradually bigger masks, taking into account the discrete nature of integral images and the specific structure of our filters.

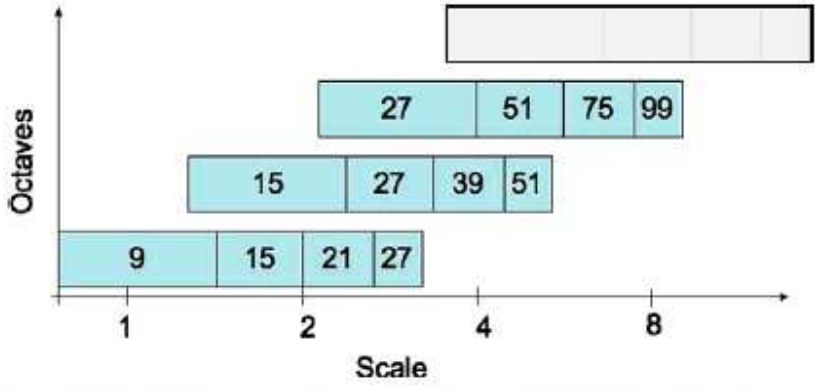


Fig. 6. Graphical representation of the filter side lengths for three different octaves. The logarithmic horizontal axis represents the scales. Note that the octaves are overlapping in order to cover all possible scales seamlessly.

The scale space is divided into octaves. An octave represents a series of filter response maps obtained by convolving the same input image with a filter of increasing size. In total, an octave encompasses a scaling factor of 2 (which implies that one needs to more than double the filter

size, see below). Each octave is subdivided into a constant number of scale levels. For keep the size uneven and thus to ensure the presence of the central pixel, the mask size is increased by 6 pixels.

3.4. Interest point Localization.

In order to localize interest points in the image and over scales, non-maximum suppression in a 3x3x3 neighborhood is applied. Specifically, we use a fast variant introduced by Neubeck and Van Gool [12]. The maxima of the determinant of the Hessian matrix are then interpolated in scale and image space with the method proposed by Brown and Lowe [5].
Scale space interpolation is especially important in our case, as the difference in scale between the first layers of every octave is relatively large.

4. INTEREST POINT DESCRIPTION AND MATCHING.

Our descriptor describes the distribution of the intensity content within the interest point neighborhood, similar to the gradient information extracted by SIFT [24] and its variants. We build on the distribution of first order Haar wavelet responses in x and y direction rather than the gradient, exploit integral images for speed, and use only 64 dimensions.

Furthermore, we present a new indexing step based on the sign of the Laplacian, which increases not only the robustness of the descriptor, but also the matching speed (by a factor of 2 in the best case). We refer to our detector-descriptor scheme as SURF - Speeded-Up Robust Features.

The first step consists of fixing a reproducible orientation based on information from a circular region around the interest point. Then, we construct a square region aligned to the selected orientation and extract the SURF descriptor from it. Finally, features are matched between two images. These three steps are explained in the following.

4.1. Orientation assignment.

In order to be invariant to image rotation, we identify a reproducible orientation for the interest points. For that purpose, we first calculate the Haar wavelet responses in x and y direction within a circular neighborhood of radius 6s around the interest point, with s the scale at which the interest point was detected.

Once the wavelet responses are calculated and weighted with a Gaussian (sigma = 2s) centered at the interest point, the responses are represented as points in a space with the horizontal response strength along the abscissa and the vertical response strength along the ordinate. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of size pi/3. The horizontal and vertical responses within the window are summed. The two summed responses then yield a local orientation vector. The longest such vector over all windows defines the orientation of the interest point. The size of the sliding window is a parameter which had to be chosen carefully. Small sizes fire on single dominating gradients; large sizes tend to yield maxima in vector length that are not outspoken. Both result in a misorientation of the interest point.

Note that for many applications, rotation invariance is not necessary. Experiments of using the upright version of SURF (U-SURF, for short) for object detection can be found in. U-SURF is

faster to compute and can increase distinctively, while maintaining robustness to rotation of about ±15°.

4.2. Descriptor based on sum of Haar wavelet responses.

For the extraction of the descriptor, the first step consists of constructing a square region centered around the interest point and oriented along the orientation selected in previous section. The size of this window is 20s.

Fig. 11. Detail of the Graffiti scene showing the size of the oriented descriptor window at different scales.

The region is split up regularly into smaller 4x4 square sub-regions. This preserves important spatial information. For each sub-region, we compute Haar wavelet responses at 5x5 regularly spaced sample points.

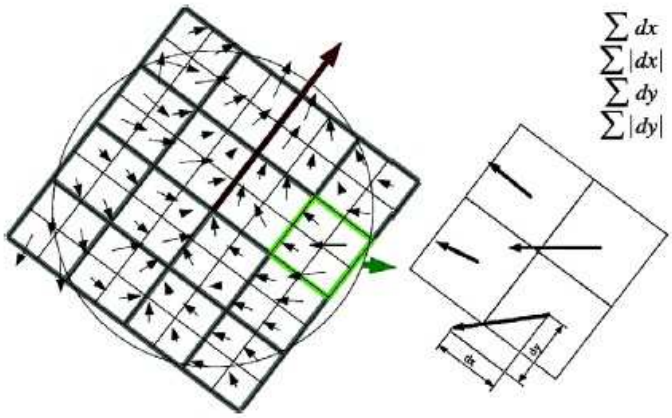


Fig. 12. To build the descriptor, an oriented quadratic grid with 4×4 square sub-regions is laid over the interest point (left). For each square, the wavelet responses are computed from 5×5 samples (for illustrative purposes, we show only 2×2 sub-divisions here). For each field, we collect the sums d_x, $|d_x|$, d_y, and $|d_y|$, computed relatively to the orientation of the grid (right).

Then, the wavelet responses dx and dy are summed up over each sub-region and form a first set of entries in the feature vector. In order to bring in information about the polarity of the intensity changes, we also extract the sum of the absolute values of the responses, |dx| and

|dy|. Hence, each sub-region has a 4D descriptor vector P for its underlying intensity structure. Concatenating this for all 4 x 4 sub regions, this results in a descriptor vector of length 64.

SURF is, up to some point, similar in concept as SIFT, in that they both focus on the spatial distribution of gradient information. Nevertheless, SURF outperforms SIFT in practically all cases, as shown in Section 5. We believe this is due to the fact that SURF integrates the gradient information within a subpatch, whereas SIFT depends on the orientations of the individual gradients.

4.3. Fast indexing for matching.

For fast indexing during the matching stage, the sign of the Laplacian (i.e. the trace of the Hessian matrix) for the underlying interest point is included. Typically, the interest points are found at blob-type structures. The sign of the Laplacian distinguishes bright blobs on dark backgrounds from the reverse situation. This feature is available at no extra computational cost as it was already computed during the detection phase. In the matching stage, we only compare features if they have the same type of contrast. Hence, this minimal information allows for faster matching, without reducing the descriptor's performance. Note that this is also of advantage for more advanced indexing methods.

TESTING

The objective is to test the performance of the library with images that have artifacts. We will discuss two cases: Artificial and natural artifacts. In the experiments we evaluate the repeatability and the number of correct matches.

State of the art

There are several papers comparing the different feature extractors. Although we analyze the performance of only one, we may use the results concerning the tests for SURF. The most common tests [2], [3], [4] are for images with Gaussian noise, variation of the point of view (angle) and change of scale. Also there is a test of pattern recognition with Blur images [4b].

The feature points and the descriptors are determined in the original image and the noisy-ones. After the matching stage, a small script is used to determine which matches are correct. The total number of correct matches is recorded for the evaluation as well the ratio of incorrect matches. This information tells us more about the quality of the implementation than the number of keypoints.

The evaluation of the feature descriptor is based in two criteria: Repeatability and Recall.

Repeatability is the ratio between the correspondences in the two images and the mean of points extracted in both images. This shows how well the geometry of the feature can be found back. A score between 0% and 100% is obtained.

$$Repeatability = \frac{Number\ of\ correspondences}{mean\ of\ detected\ points\ in\ both\ images}$$

Recall helps to evaluate the descriptor. Again a score between 0 and 100 results where 0 is the best grade. It takes into account not only the match with the descriptor criteria, also considers the geometrical correspondence. If a match is above a threshold, it is considered as a bad match.

$$Recall = \frac{Incorrect\ matches}{Total\ matches}$$

Due to the most of the information just compared the different feature extraction algorithms, in all of them is remarked the fact that SURF is the faster one and the one who extracts less points. In general the performance is similar to SIFT but a bit worse.
The rotation angle does not influence much in the correct matches, as well for the scale changes. But for the noisy or blur is hardly dependent, losing performance when the presence of noise is high. Change the viewpoint angle drop the performance after changes of 10 degrees. Illumination changes are not critical up to a certain level, when not enough keypoints are generated.

Artificial artifacts.

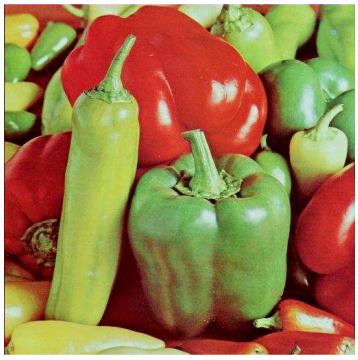
In this part we test the performance of SURF in presence of noise (Gaussian and Poisson), brightness and darkness, movement and blur. For all the cases, an image is read and it is generated other image with the same size, but with the modifications.

The evaluation of the feature descriptor is based in two criteria: Repeatability and Recall. These factors were already explained in state of the art part.

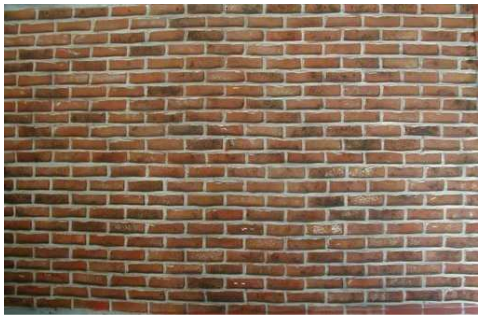
Test images:

Lenna

Sunflowers



Peppers



Brick wall

Gaussian noise

Gaussian noise of mean 0 and standard deviation between 0 and 0.1 is added to the second image. So if the first image is I1, the second is I2 = I1 + G, where G is the noise. The histogram of G will be a Gaussian distribution with mean 0 and standard deviation sigma (which varies). Since we are working with random numbers, we repeat the experiment for each standard deviation, and we take as a result the mean of Repeatability and Recall.

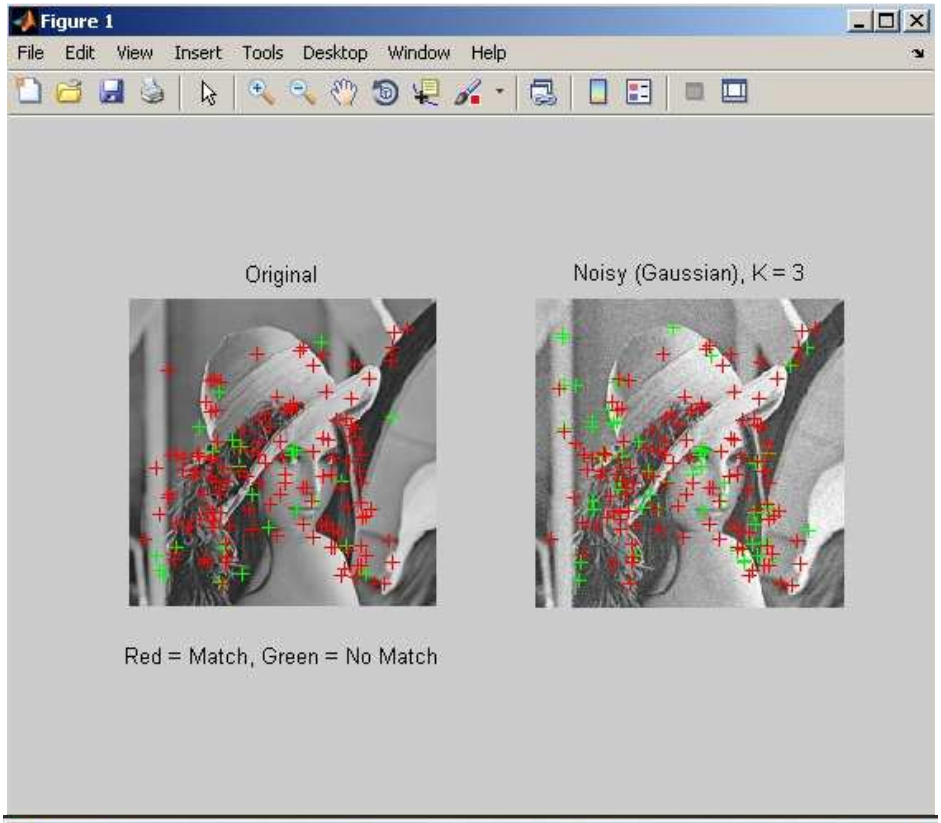


Figure 1
File Edit View Insert Tools Desktop Window Help
Original
Noisy (Gaussian), K = 3
Red = Match, Green = No Match

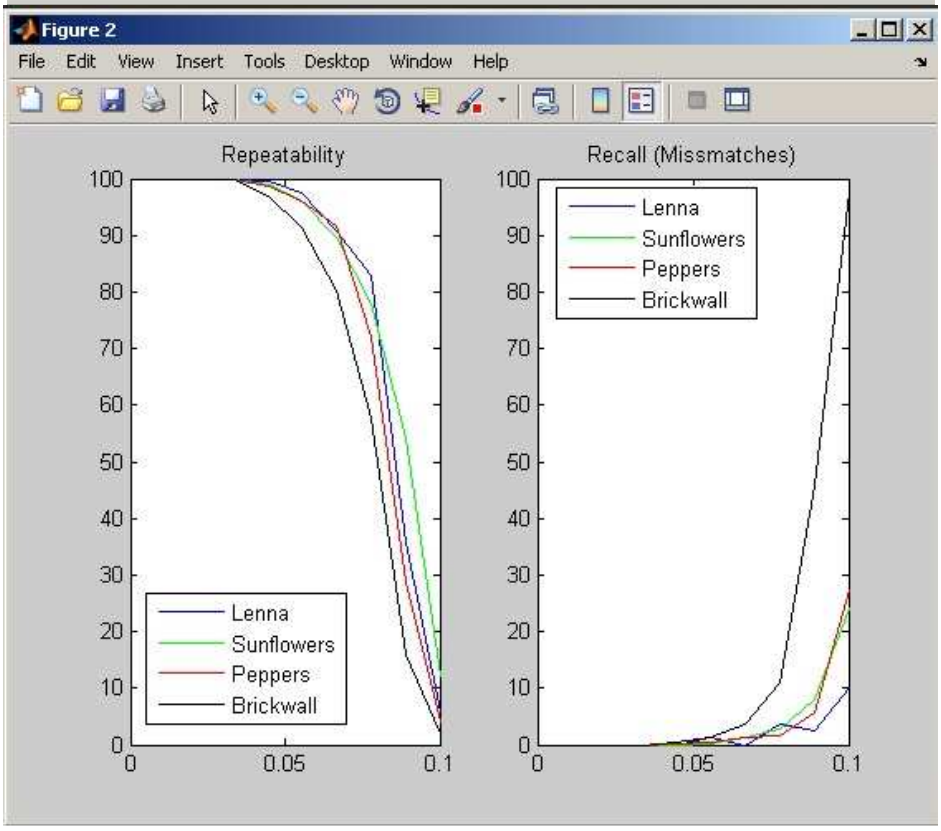
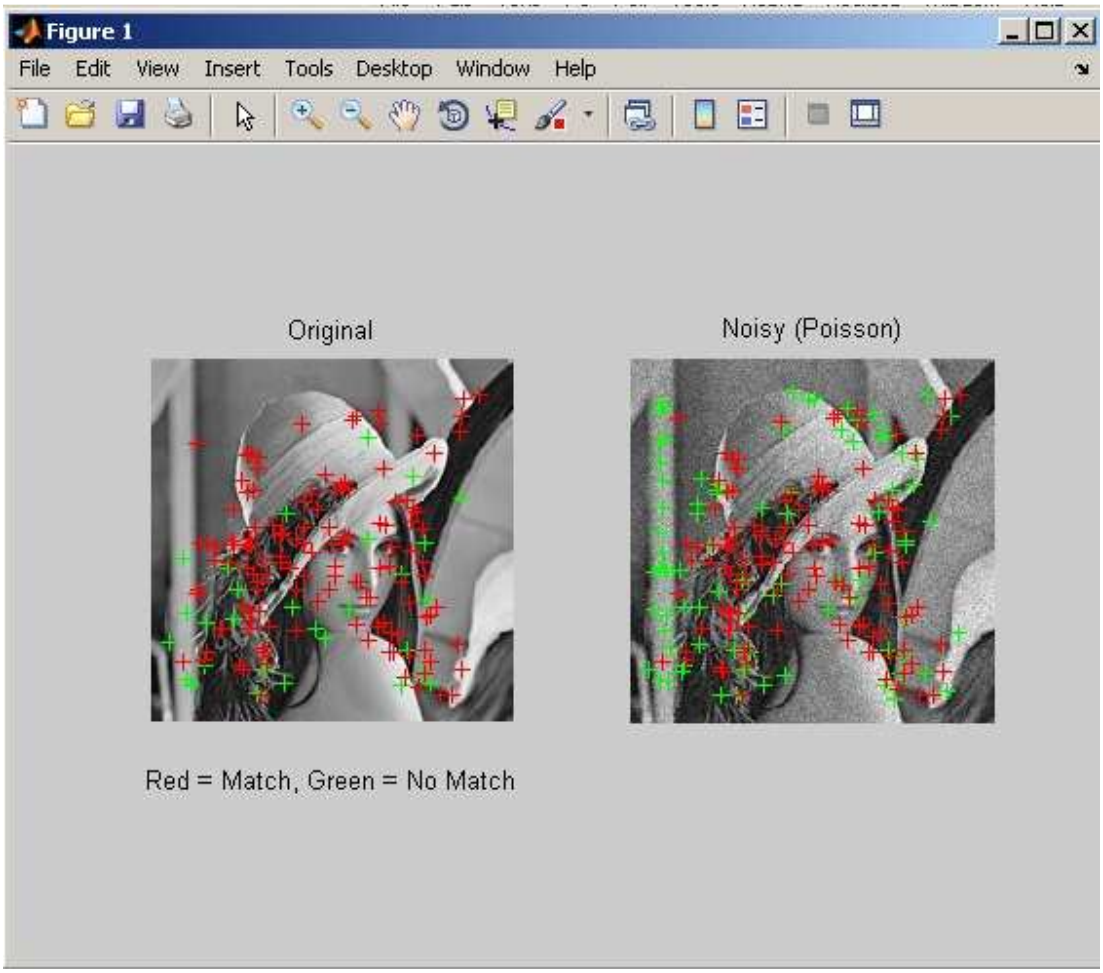


Figure 2
File Edit View Insert Tools Desktop Window Help
Repeatability
Recall (Missmatches)
Lenna
Sunflowers
Peppers
Brickwall
0
10
20
30
40
50
60
70
80
90
100
0.05
0.1

We can see the there are two regions. In the first one, the loss of performance is low until a certain value. After that value, the loss of performance is faster. We may consider that SURF is not very sensitive to Gaussian noise, except when it reaches a level when the Repeatability is low but the Recall it is still in an acceptable level, with the exception of Brickwall. Therefore we may conclude the algorithm is robust to Gaussian noise.

Poisson noise

For the Possion noise, the procedure is the same to the previous case but now I2 will be: I2 = I1 + P, where P is the Poisson noise. The distribution is centered at 0, and the other parameters depend on the input image.

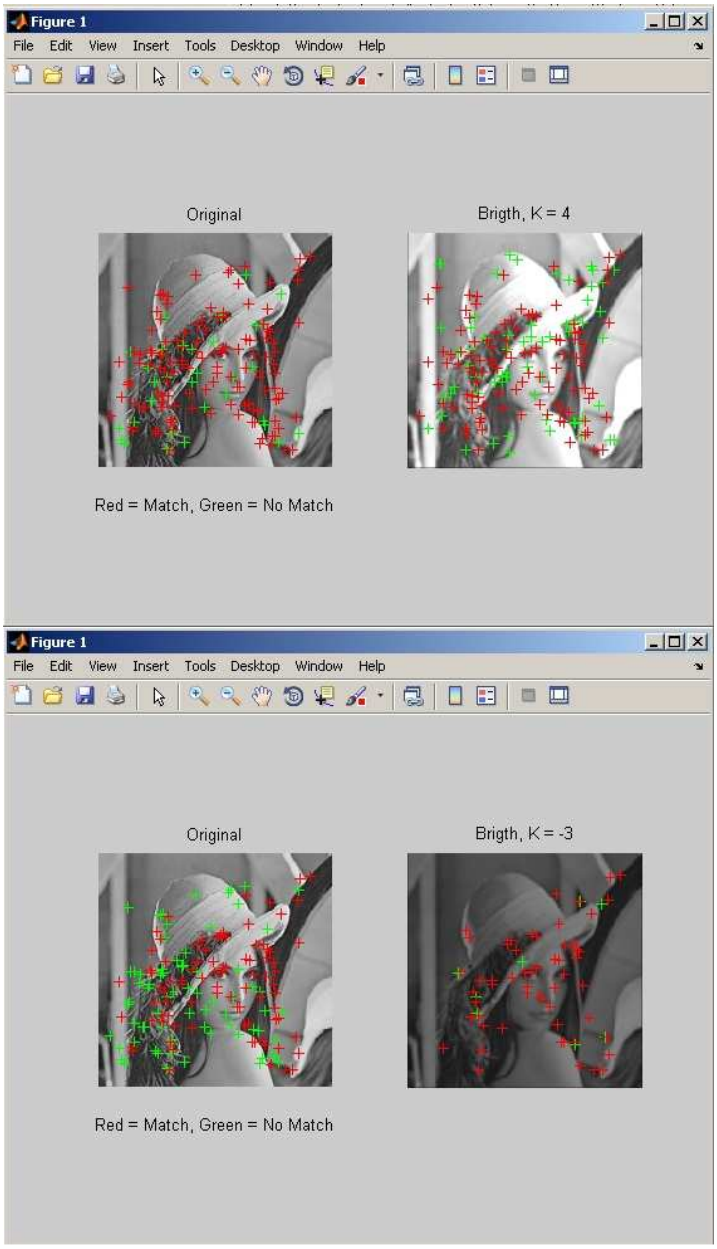


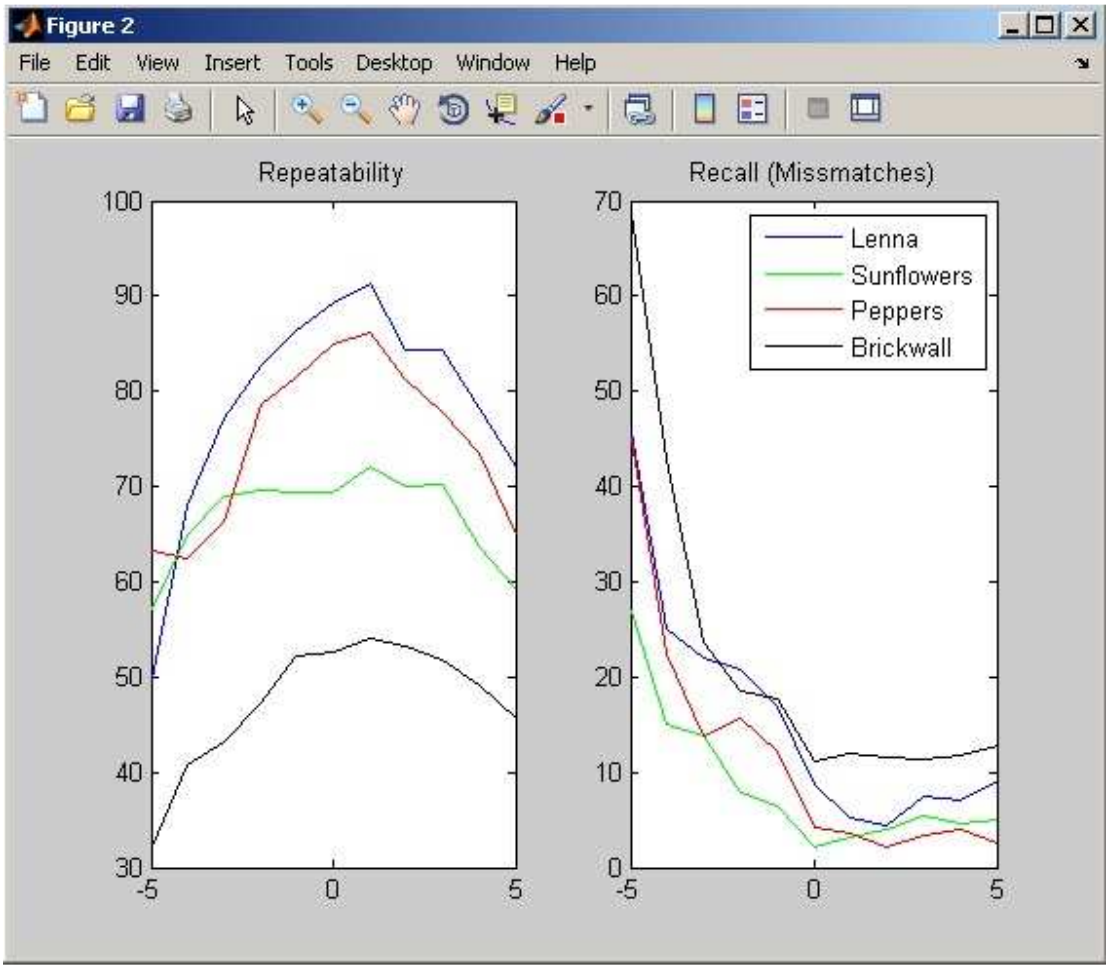
	Lenna	Sunflowers	Peppers	Brick wall
Ratio of nomatches	70.93%	73.1193%	66.9106%	48.7569%
Ratio of incorrect matches	0.0075%	0.971%	1.879%	8.8677%

We may consider that SURF is robust to Poisson noise, with an accurate matching.

Brightness variations

For this case the second image is convoluted with a mask of varying mean: Greater than 1 for have a shinier image, and lower than 1 for a darker image having at k=0 a mask of mean equal to 1. So I2 = conv(I1, mask). The mask is a matrix of ones (3x3), and divided with a factor (which changes). We begin with the darkest image (k=-5) and we change the value up to k=5 (the shiniest image). The mean changes in both directions the same amount, so for k=-5 the mean is 0.4 and for k=5 the mean is 1.6.





SURF is sensitive to the changes of brightness, more for the darkness than to the brightness. Still, with the Repeatability there is not a big difference in both regions, but with the Recall there is a different behavior between dark and bright, having the worst performance in the dark region. Furthermore we may consider two parts, one when the loss of performance is slow, and other when the loss of performance is fast, as we have seen in the state of the art part.

Motion blur

In this part a filter is created that after the convolution it is approximated a linear motion of a selected length and angle. The function fspecial is used to generate a mask that gives the effect of motion blur. The angle and the length of the movement are varied to check if there is dependence not only in the quantity of movement, but also the angle.

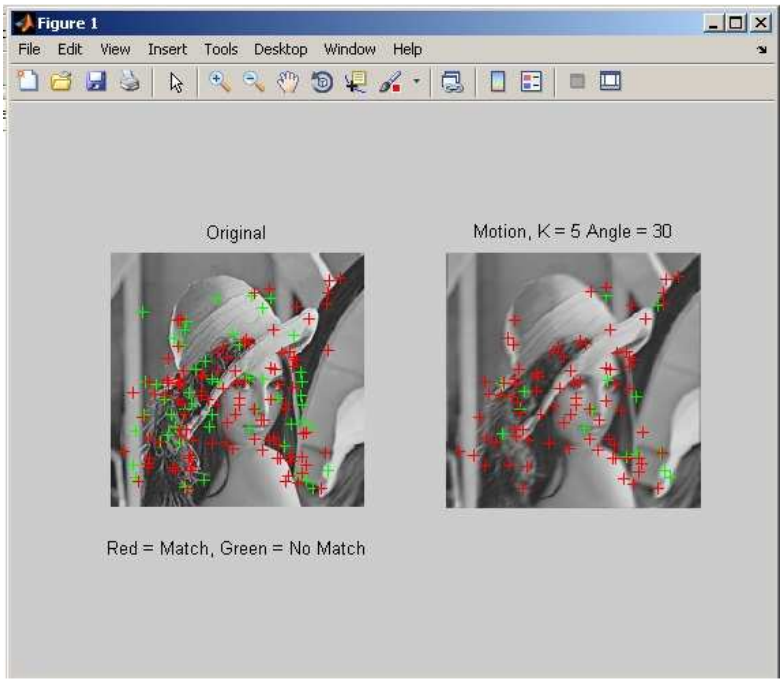


Figure 1
File Edit View Insert Tools Desktop Window Help
Original
Motion, K = 5 Angle = 30
Red = Match, Green = No Match

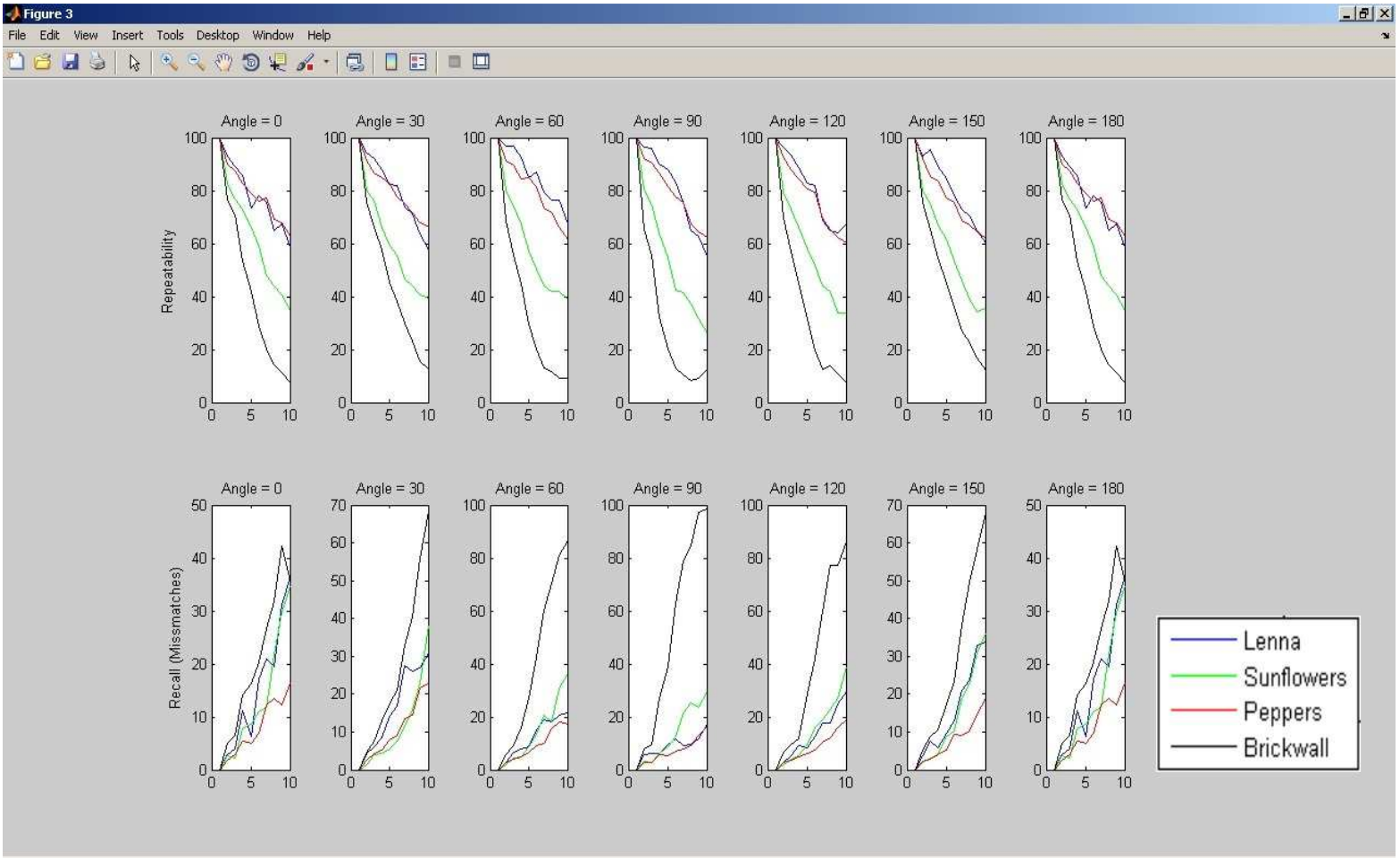
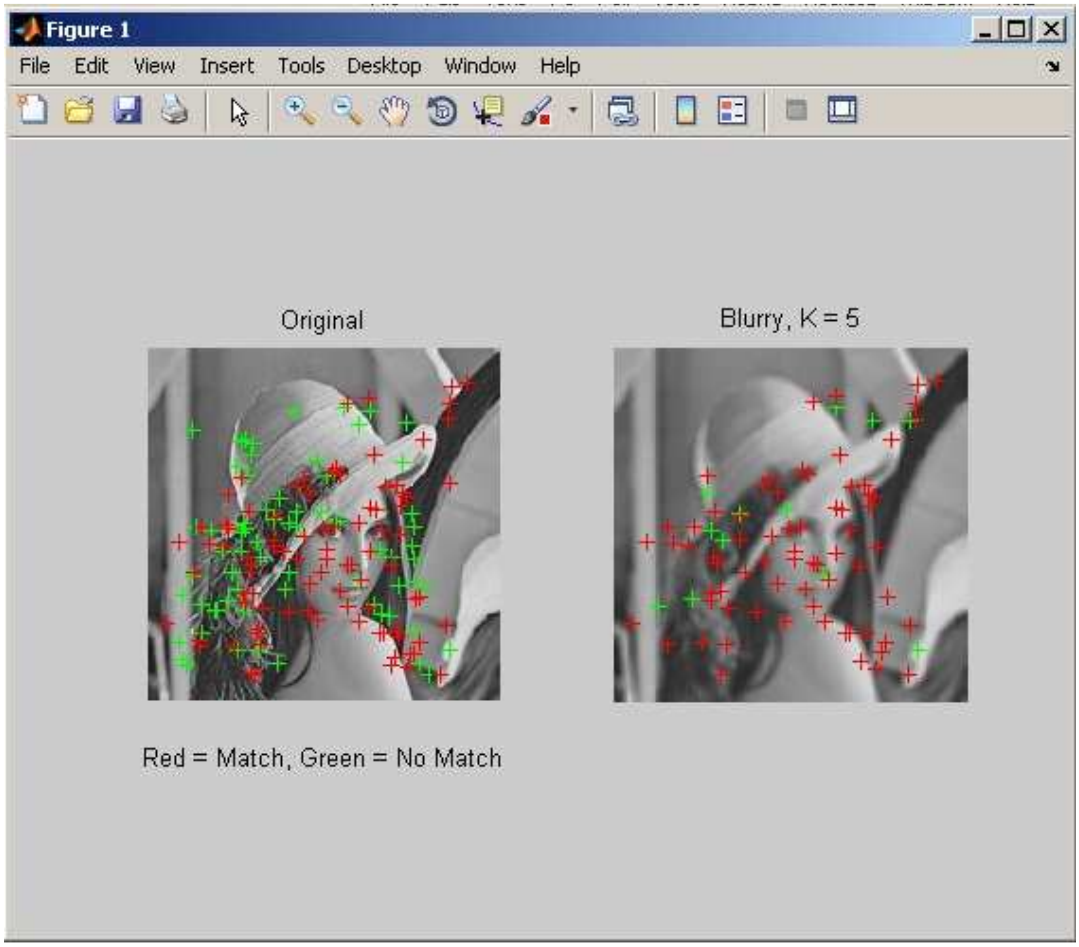


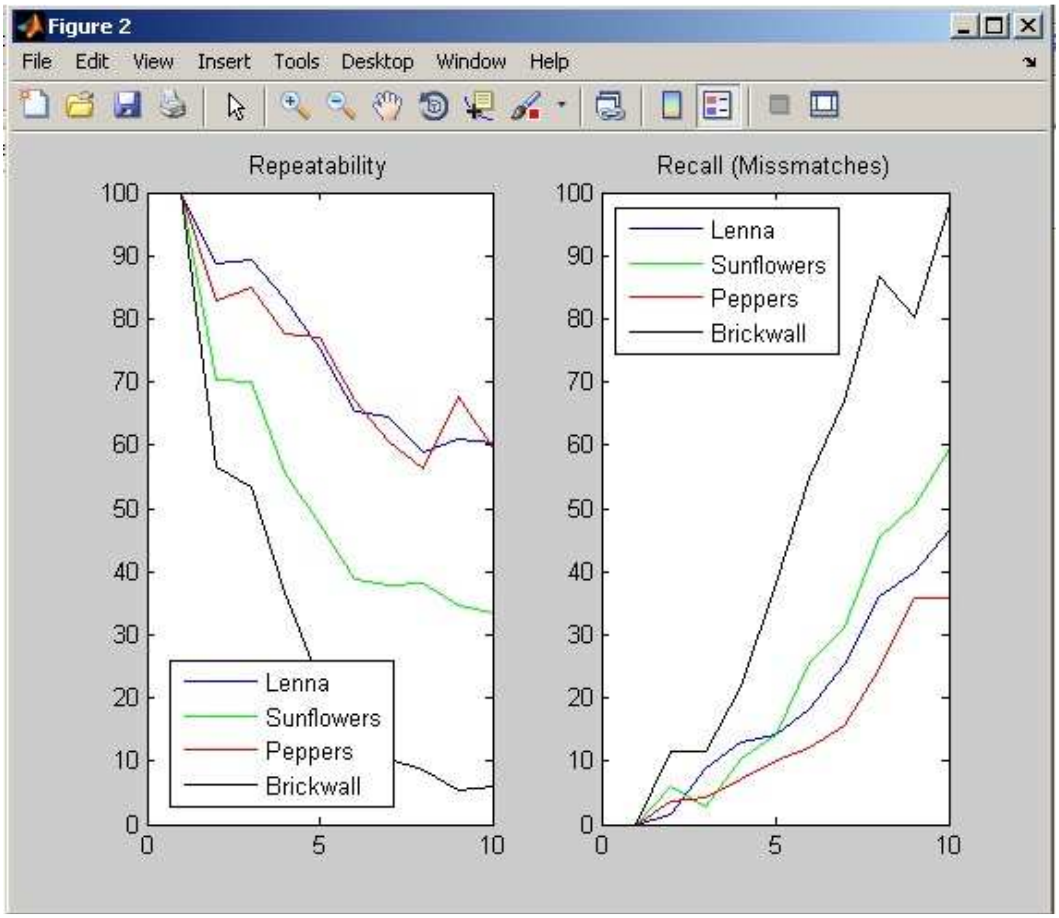
Figure 3
File Edit View Insert Tools Desktop Window Help
Angle = 0
Angle = 30
Angle = 60
Angle = 90
Angle = 120
Angle = 150
Angle = 180
Repeatability
Recall (Missmatches)
Lenna
Sunflowers
Peppers
Brickwall

About the dependence between loss of performance and length of movement, in general is regular, with some changes depending on the angle. The same about positive or negative displacement (look 0 and 180 degrees). About the angle of displacement, in the Repeatability part there are not many changes, and in the Recall part there is a slightly better performance in the second quadrant (90 to 180 degrees). Also is noticeable that the presence of lines or a characteristic pattern with an orientation, it is strongly affected the performance if the displacement is in a perpendicular direction to the orientation.

Blur (out of focus blur)

An average filter of increasing size is applied to the image, obtaining a blurry image. So I2 = imfilter(I1, mask). The average filter is created with the function fspecial, which is a matrix of ones divided by a factor (for make the mean equal to one). The convolution is defined by:
z = y*h = y1h9 + y2h8 + y3h7 + y4h6 + y5h5 + y6h4 + y7h3 + y8h2 + y9h1, being y and h 3x3 matrices.





The loss of performance is constant. Still, more than 50% of the matches are correct but the repeatability is low at high levels of blurry. The exception is Brickwall because at certain levels of blurry the image looks uniform and the characteristic pattern is lost. We can see that SURF is very sensitive to blur.

General conclusion

SURF has in general a good tolerance to noise. The performance is acceptable up to a certain level of noise when there is a loss of performance. The same we may say for the changes of brightness, especially for shiny images. The contrary we observed with darker images since repeatability is low and the Recall is lower. For Blur we conclude that SURF shows bad results because fewer points are matched and more matches are wrong, highly affected by the morphology of the image.

Nevertheless, the image pattern affects much the performance. An image with uniform pattern will have a poor performance (Brickwall), especially in some cases if the pattern has an orientation (a grid for example). The same if it has a pattern that is repeated in the image (Sunflowers)

Natural artifacts

In this part we use raw data, varying the options in the camera. Now the artifacts will appear as a result of the change in the parameters of the camera, as well as the defects of the camera.

Types of noise

The image noise is a random variation in the brightness or color produced by the sensor and circuitry of the digital camera. It is an undesirable product because it gives a less accurate portrayal of the subject.

The standard model of amplifier noise is additive, Gaussian, independent at each pixel and independent of the signal intensity, caused primarily by thermal noise, including that which comes from the reset noise of capacitors. In color cameras where more amplification is used in the blue color channel than in the green or red channel, there can be more noise in the red channel. Gaussian noise is a major part of the "read noise" of an image sensor, that is, of the constant noise level in dark areas of the image.

Image with salt and pepper noise.

Other type of noise is the sometimes called salt-and-pepper noise or spike noise. An image containing salt-and-pepper noise will have dark pixels in bright regions and bright pixels in dark regions. This type of noise can be caused by dead pixels, analog-to-digital converter errors, bit errors in transmission, etc.

The dominant noise in the lighter parts of an image from an image sensor is typically that caused by statistical quantum fluctuations, that is, variation in the number of photons sensed at a given exposure level; this noise is known as photon shot noise. Shot noise has a root-mean-square value proportional to the square root of the image intensity, and the noises at different pixels are independent of one another. Shot noise follows a Poisson distribution, which is usually not very different from Gaussian.

In addition to photon shot noise, there can be additional shot noise from the dark leakage current in the image sensor; this noise is sometimes known as "dark shot noise" or "dark-current shot noise". Dark current is greatest at "hot pixels" within the image sensor; the variable dark charge of normal and hot pixels can be subtracted off (using "dark frame subtraction"), leaving only the shot noise, or random component, of the leakage; if dark-frame subtraction is not done, or if the exposure time is long enough that the hot pixel charge exceeds the linear charge capacity, the noise will be more than just shot noise, and hot pixels appear as salt-and-pepper noise.

The noise caused by quantizing the pixels of a sensed image to a number of discrete levels is known as quantization noise; it has an approximately uniform distribution, and can be signal dependent, though it will be signal independent if other noise sources are big enough to cause dithering, or if dithering is explicitly applied.

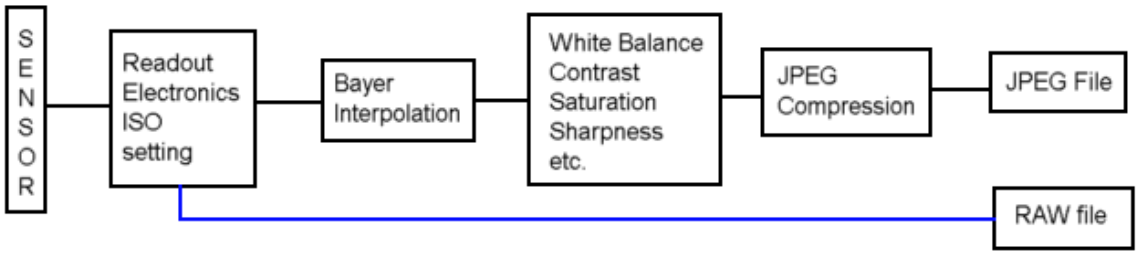
In low light, correct exposure requires the use of long shutter speeds, higher gain (ISO sensitivity), or both. On most cameras, longer shutter speeds lead to increased salt-and-pepper noise due to photodiode leakage currents. At the cost of a doubling of read noise variance (41% increases in read noise standard deviation). The relative effect of both read noise and shot noise increase as the exposure is reduced, corresponding to increased ISO sensitivity, since fewer photons are counted (shot noise) and since more amplification of the signal is necessary.

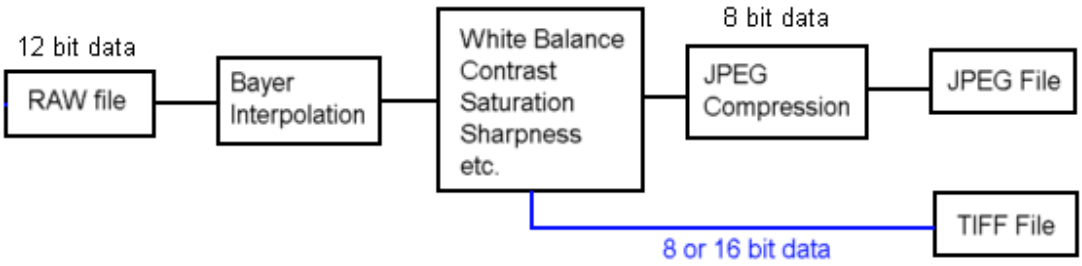
State of the art

The literature presents no study concerning the performance of SURF with raw images.

RAW data

A camera raw image file contains minimally processed data from the image sensor. Raw files are so named because they are not yet processed and therefore are not ready to be printed or edited with a bitmap graphics editor.
Raw image files are sometimes called digital negatives, as they fulfill the same role as negatives in film photography: that is, the negative is not directly usable as an image, but has all of the information needed to create an image. The purpose of raw image formats is to save, with minimum loss of information, data obtained from the sensor, and the conditions surrounding the capturing of the image.

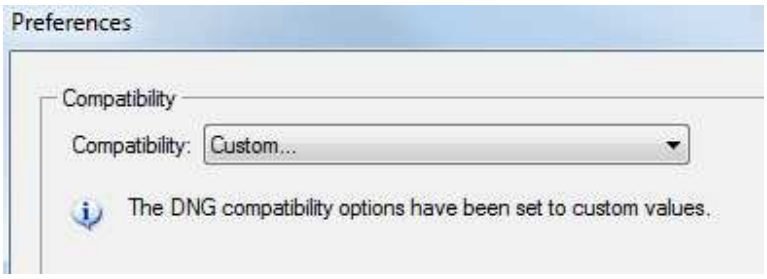


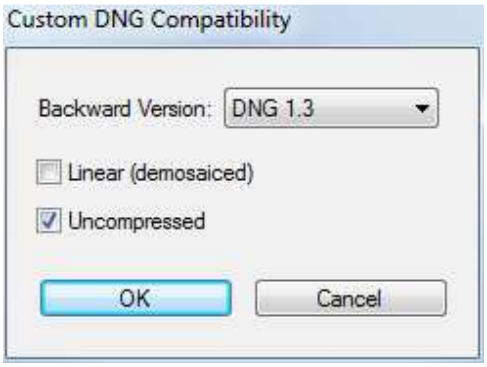


These images are often described as "*RAW* image files", although there is not actually one single raw file format. In fact there are dozens if not hundreds of such formats in use by different models of digital equipment (like cameras or film scanners). Because of that, we select the .DNG format for work, and we use the *Adobe image converter* for transform the other types of raw files to .DNG. Also, for help with working with the raw files, we use the *FastStone image viewer* for have a preview of the images.

Raw data and Matlab

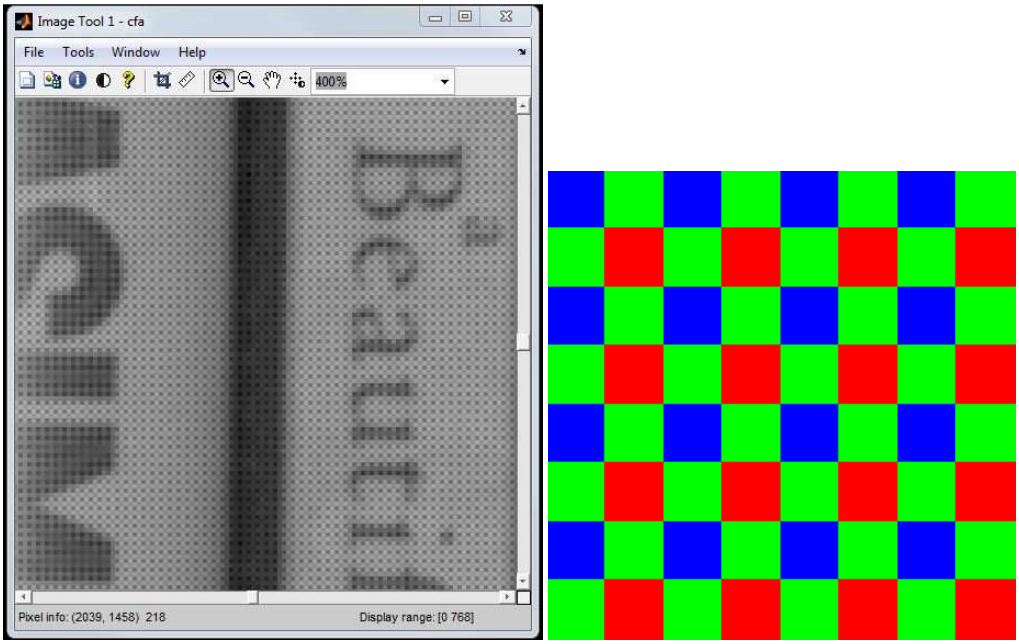
First of all due to the big amount of proprietary formats for raw files, we decide to use DNG since is the format that Matlab works with better. For convert the files to DNG, Adobe DNG converter is used. In the reference [13] it is explained that there are some issues that after some trial and error are solved. The options that must be chosen for the converter are shown in the following images:





After we do the conversion we can begin to work with the files. The raw file is treated as a normal file. More precisely a raw file can be treated as a TIFF file [14], for be able to view the image and process it. Actually the use of imread or imshow functions will not give a real result, it will just show the thumbnail image. The data we need is in the *sub image file directory tag*.

Then, we create a Tiff object with the raw file, we get the subIFD tag, and after set the offset we are ready to work with the *color filter array* image. But before we use the SURF functions, it is necessary to separate the color channels and transform the image to an 8-bit image.



Example of visualization. It is noticeable the pattern of the Bayer filter (right)

Procedure

The starting point is the CFA (Color Format Array) image. The first thing is use a loop for get the pixels of each color. For the green channel, since there are two channels, we use both values. For the reference image we average the green channels, and we scale the images dividing each one by its mean. This it is done for have the same mean and also because the ISO factor does not work 100% perfectly, and we do the scaling for correct it.

Experiment 1

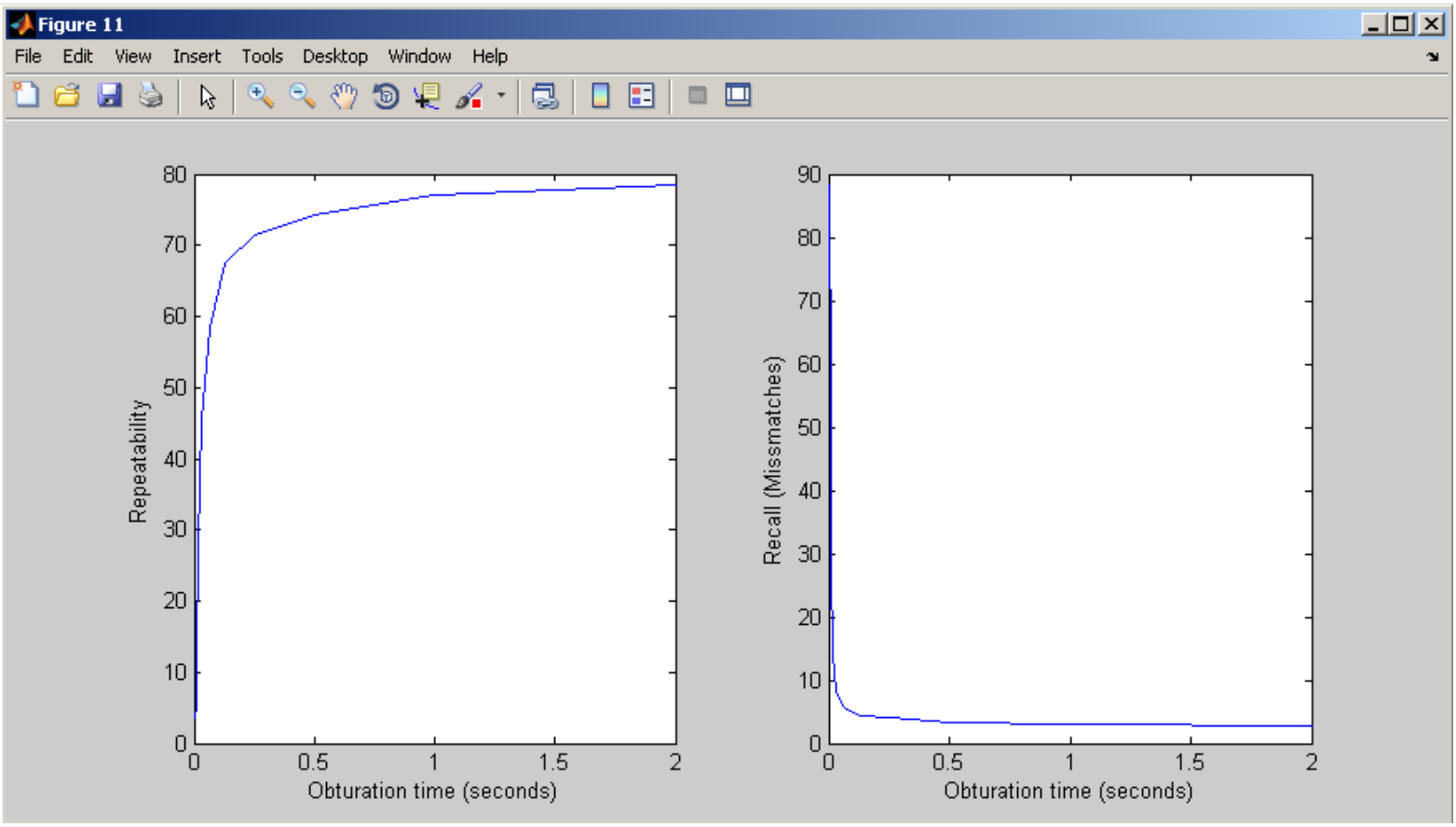
For the experiment a proper scene is chosen. Was selected a desktop, avoiding regular patterns and trying to have a variety of contrast, for do not saturate the sensor and have a good performance of SURF with no noisy images. Since we evaluate random data, we take 5 pictures for each exposure time, and then we average the results. The parameters that we will evaluate are repeatability and Recall, explained in the previous part.

We start with an exposure time of 2 seconds and an ISO of 100. Then we divide by two the exposure time and we multiply by two the ISO. The combinations are shown in the following table:

Time	2’	1’	1/2’	1/4’	1/8’	1/15’	1/30’	1/60’	1/125’	1/250’	1/500’
ISO	100	200	400	800	1600	1600	1600	1600	1600	1600	1600

It will be noticeable that with the short exposure times will appear random noise, and the image will be darker. The scene has to be chosen carefully in order to do not saturate the sensor with high exposure times.

Examples of images with different exposure times (the right one has shorter exposure time)

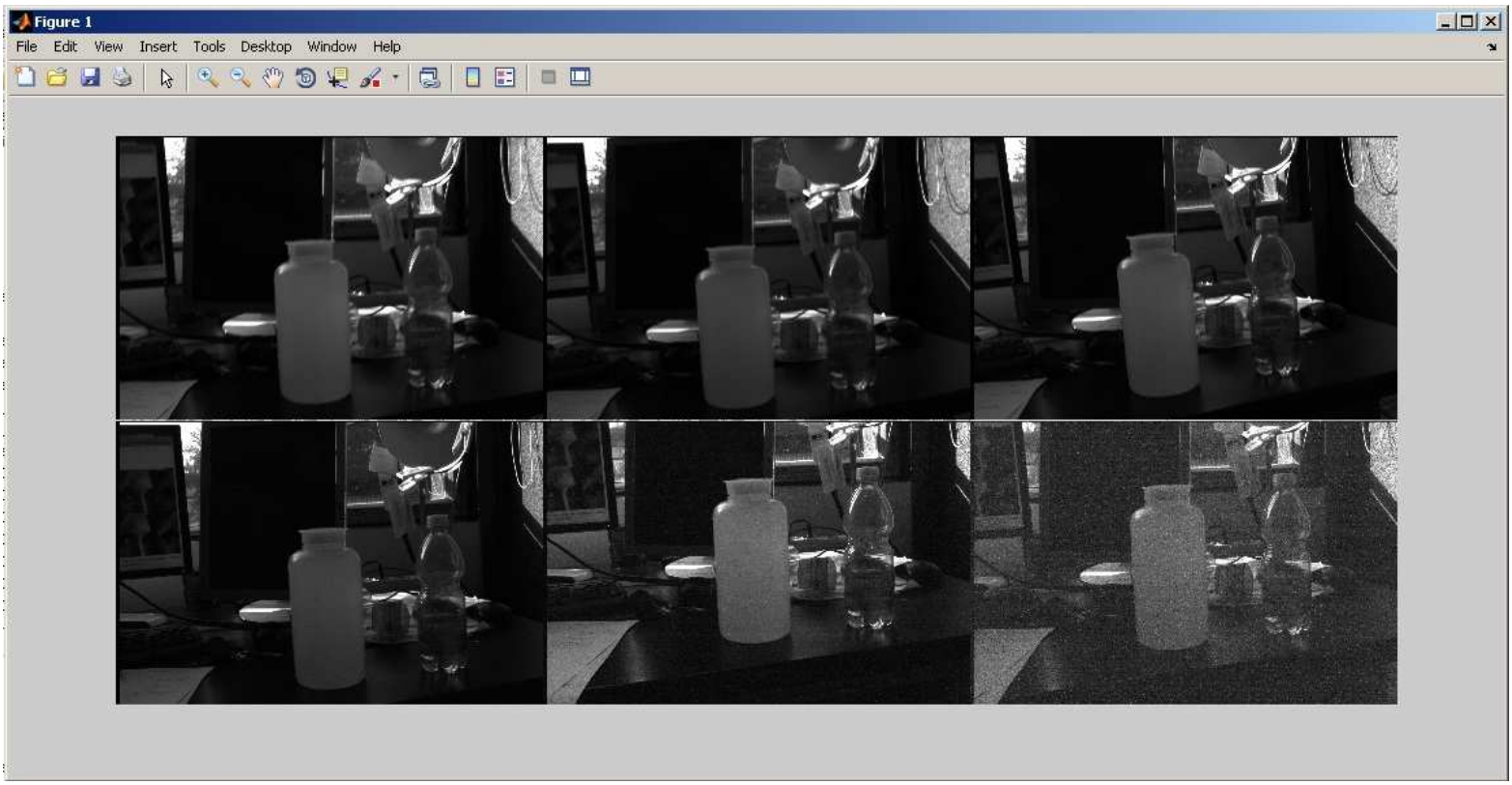


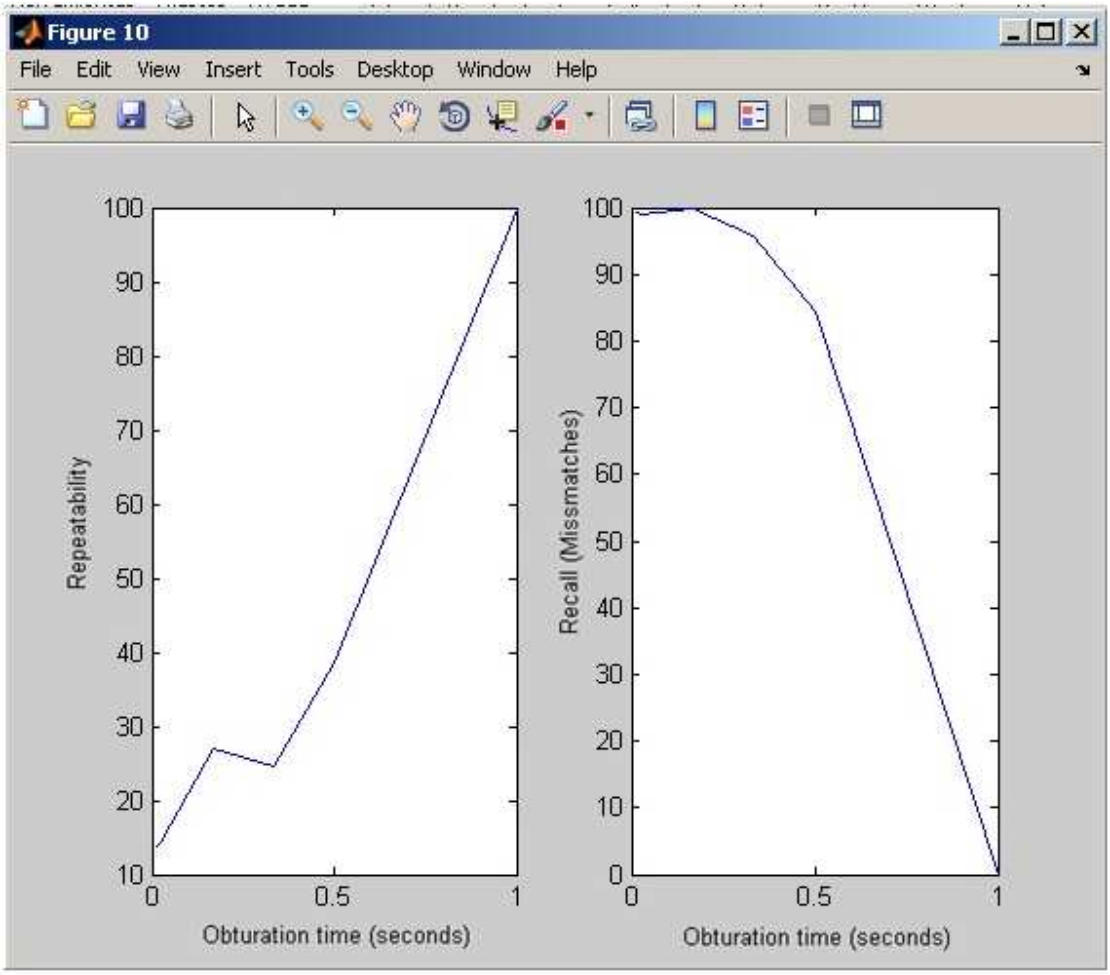
We may see that when the exposure time is short the performance is bad. This is due to the darkening of the image and the noise associated to the shooting (shot noise). Also we see that the performance decay very quickly if the exposure time is below 1/15'. Nevertheless, the performance is not perfect in the best case because there is more noise than in a processed image. We may conclude SURF is robust to natural noise as well.

Other experiments (less rigorous)

Experiment 2

These photos were taken in the office for begin to work. The exposure time is decreasing, being the last picture the one with the shortest exposure time. In comparation with the previous experiment, the scene has more light and as a consequence the short exposure images will be noisier.



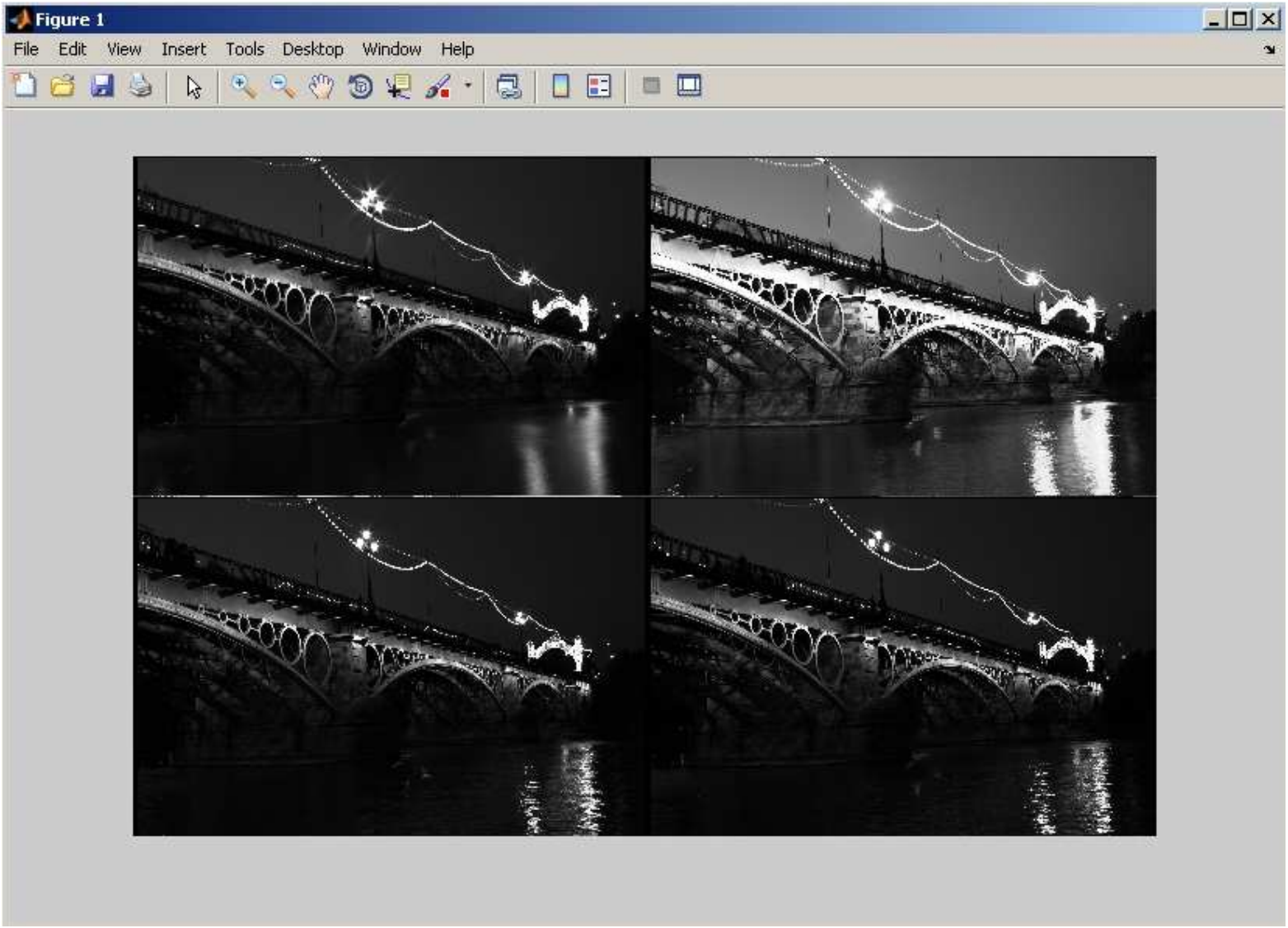


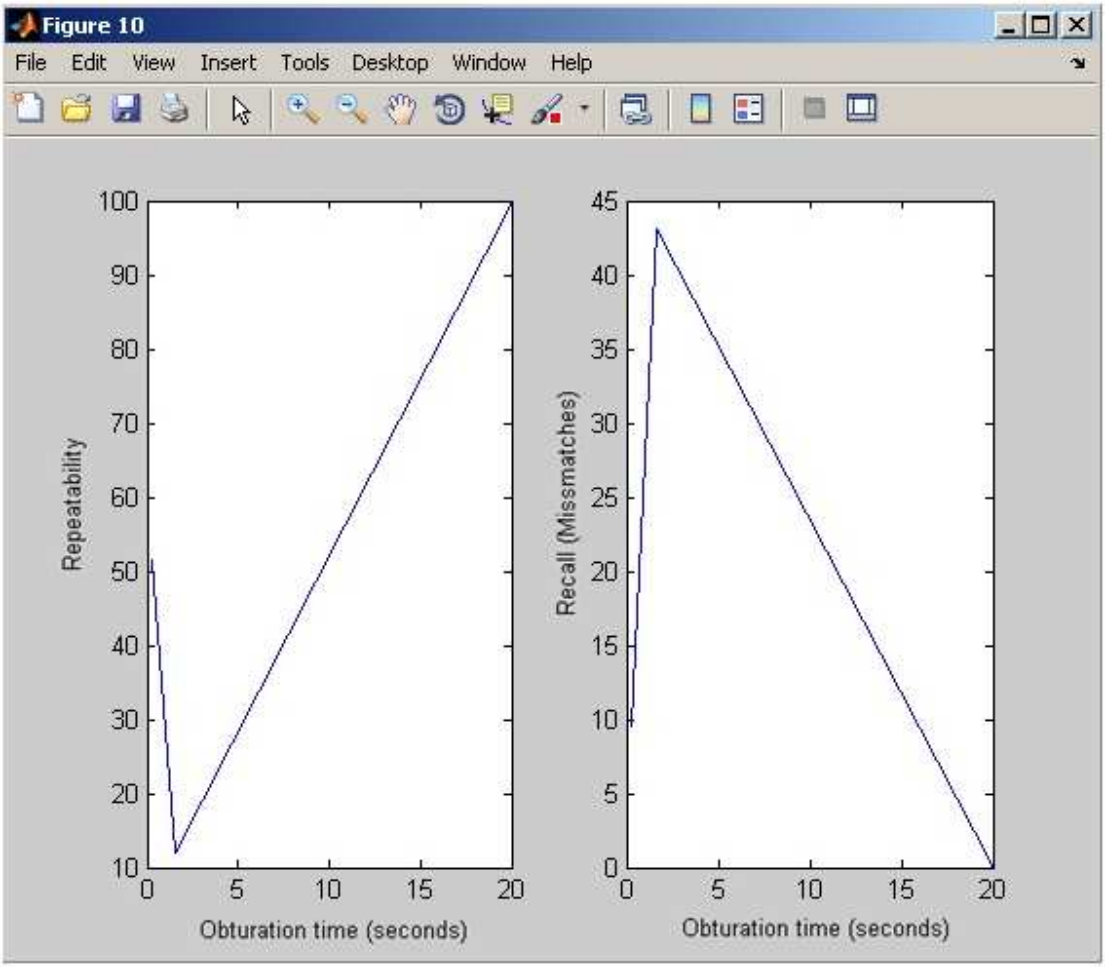
In first place we must say that the pictures are not exactly the same, since the camera is not in a fixed base and the landscape changes from one picture to the other so even with no noise the results could not be perfect. We can see that in the three first pictures, when there is no significative presence of noise but the performance decay quickly.
Apart of that issue, we may consider that the performance is much worse than with processed pictures since there is more noise and artifacts in these cases, up to a level we consider SURF does not do a good matching. But these results cannot be taken as good results because only one picture was taken for each exposure time.

Experiment 3

These pictures were obtained from somebody that gave me.

In this case also the exposure time is changing, as in the previous case. The camera is in a tripod and the maximum exposure time is very long.





Here we do not have the problems of the fixed base since the camera is in a tripod, so the changes are only those of exposure time. Also one picture was taken for each exposure time. Still we may consider that SURF is sensible to this parameter, more when it is short.

Problems

When working with an image when there is a frame with no keypoints (blank scene with all white pixels, or low illumination, and no features), the program crashes with error message:

??? Unexpected Standard exception from MEX file.
What() is:..\..\..\src\cxcore\cxarray.cpp:113: error: (-201) Non-positive width or height

The solution consists to edit surf.cpp [15,16] but since we are using a mex file, and there is not available the surfpoints.cpp for edit, we cannot solve this problem. A solution consists in adjust the image histogram. A good solution is use the function imadjust, with enhances the contrast and solves the problem. But a better solution is to avoid take pictures with very short exposure time or avoid dark scenes.

APPENDIX. PORTING SURF LIBRARIES TO MATLAB

Introduction

MATLAB has the capability of running functions written in C. The files which hold the source for these functions are called MEX-Files [18]. The mex Functions are not intended to be a substitute for MATLAB's Built-In operations however if you need to code many loops and other things that MATLAB is not very good at, this is a good option. This feature also allows system-specific APIs to be called to extend MATLAB's abilities.

Making possible the use of mex files with Visual studio 2010 (add compiler)

Configure the compiler:

The first thing to do is to configure the compiler. For do that we introduce the command *mex -setup* and we simply choose it from the list [19]. For compile SURF we use the compiler of Microsoft Visual Studio 2010. In the case it does not appear the most probable reason is because if Visual Studio is in another language than English Matlab wont detect it [17]. For solve this path has to be specified manually. In addition, for VS 2010 we need to put a patch for work with it [20,21].

```
>> mex -setup
Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\PROGRA~1\MATLAB\R2010a\sys\lcc
[2] Microsoft Visual C++ 2010 in c:\Program Files\Microsoft Visual Studio 10.0

[0] None

Compiler: 2

Please verify your choices:

Compiler: Microsoft Visual C++ 2010
Location: c:\Program Files\Microsoft Visual Studio 10.0

Are these correct [y]/n? y

************************************************************************
  Warning: MEX-files generated using Microsoft Visual C++ 2010 require
           that Microsoft Visual Studio 2010 run-time libraries be
           available on the computer they are run on.
           If you plan to redistribute your MEX-files to other MATLAB
           users, be sure that they have the run-time libraries.
************************************************************************

Trying to update options file: C:\Users\Manuel\AppData\Roaming\MathWorks\MATLAB\R2010a\mexopts.bat
From template:              C:\PROGRA~1\MATLAB\R2010a\bin\win32\mexopts\msvc100opts.bat

Done . . .
```

The MEX-Function: Interface to MATLAB

The main work is based on the paper published by Rachely Esman and Yoad Snapir [22], which explains the port of OpenCV in Matlab. We use that work as a starting point for add SURF. After add the compiler, the file mexopts.bat is modified for include the OpenCV libraries in the library path. the file mexopts.bat can be easily found with the command: fullfile(prefdir,'mexopts.bat'). Also some tutorials and examples where used to add OpenCV in Matlab and begin to work with it [23,24,25,26].

After the port is done, we only need to write a short program in C that uses SURF, opening the images, computing the points and matching them [31,32,33]. Furthermore, for the code several sources are consulted, such as the OpenCV wiki, tutorials and examples [34]. In the process write the code there may be several kinds of errors:

Link error:
The error received is a linker error which indicates that the C++ source file that is being MEXed does not contain the MEX gateway function "mexFunction". To create a MEX file using source code that is present in multiple source files, provide all the source files as input to the MEX function [27].
Char16 error:
The solution consists to use namespaces in the includes [28]
Specified module could not be found:
This problem is easily solved by putting dll files n the DOS path or in the same folder as the MEX-file [29,30].
error LNK2001: unresolved external symbol mexFunction
C:\USERS\MANUEL\APPDATA\LOCAL\TEMP\MEX_CF~1\templib.x
The error you are getting means that you have not included the gateway mexFunction interface to the code (C:\Depends\depends.exe showImage.mexw32).

After the C program is done, it needs to be compiled. This is done by simply writing mex surfpoints.cpp and Matlab will generate a mex file which is the function that will do the same as the program in C, but in the Matlab environment.
Petter Strandmark did a port using the procedure explained above [35]. It has three functions: surfpoints.m, surfmatch.m and surfplot.m. The first computes the points, the second matches the points and the third plots them. It is also implemented in a way that the parameters can be modified by the structure surfoptions. For the ease of use and good results, this port it is used for all the experiments.

References

[1] Speeded-Up Robust Features (SURF), Herbert Bay, Andreas Ess, Tinne Tuytelaars and Luc Van Gool (2008)
[2] Martin L. Wyss, “Robustness of diffrent feature extraction methods against image compression”
[3] Johannes Bauer, Niko S¨underhauf, Peter Protzel, “COMPARING SEVERAL IMPLEMENTATIONS OF TWO RECENTLY PUBLISHED FEATURE DETECTORS”
[4] Oubong Gwun, Luo Juan,“A Comparison of SIFT, PCA-SIFT and SURF”
[4b] IMAGE-GUIDED TOURS: FAST-APPROXIMATED SIFT WITH U-SURF FEATURES
Eric Chu, Erin Hsu, Sandy Yu, Department of Electrical Engineering, Stanford University
[5]. 27. Brown, M., Lowe, D.: Invariant features from interest point groups. In: BMVC. (2002)
[6]. Harris, C., Stephens, M.: A combined corner and edge detector. In: Proceedings of the Alvey Vision Conference. (1988) 147 – 151
[7]. Lindeberg, T.: Feature detection with automatic scale selection. IJCV 30(2) (1998) 79 – 116
[8]. Lowe, D.: Distinctive image features from scale-invariant keypoints, cascade filtering approach. IJCV 60 (2004) 91 – 110
[9]. Mikolajczyk, K., Schmid, C.: Indexing based on scale invariant interest points. In: ICCV. Volume 1. (2001) 525 – 531
[10]. Mikolajczyk, K., Schmid, C.: Scale and affine invariant interest point detectors. IJCV 60 (2004) 63 – 86
[11]. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. PAMI 27 (2005) 1615–1630
[12]. A. Neubeck and L. Van Gool. Efficient non-maximum suppression. In ICPR, 2006
[13] http://blogs.mathworks.com/steve/2011/03/08/tips-for-reading-a-camera-raw-file-into-matlab/
[14] http://www.mathworks.es/help/matlab/ref/tiffclass.html
[15] https://code.ros.org/trac/opencv/ticket/373
[16] https://code.ros.org/trac/opencv/ticket/68
[17] http://www.mathworks.com/matlabcentral/newsreader/view_thread/297050
[18] http://www.mathworks.com/support/tech-notes/1600/1605.html
[19]http://newsgroups.derkeiler.com/Archive/Comp/comp.soft-sys.matlab/2008-04/msg03046.html
[20] http://www.mathworks.com/matlabcentral/newsreader/view_thread/279872
[21] http://www.mathworks.com/support/solutions/en/data/1-D5W493/?solution=1-D5W493
[22]http://www.mathworks.com/matlabcentral/fx_files/21818/1/OpenCV_And_MEX_Files_quick_guide.pdf
[23] http://cnx.org/content/m12348/latest/
[24] http://whatevericode.wordpress.com/2008/11/13/connecting-opencv-with-matlab-basic/
[25] http://www.roman10.net/?p=131
[26] http://sprockwell-tech.blogspot.com/2010/05/opencv-and-matlab-and-visual-studio-c.html
[27] http://www.mathworks.com/support/solutions/en/data/1-BQTPVJ/index.html?product=ML&solution=1-BQTPVJ
[28]http://www.mathworks.com/matlabcentral/newsreader/view_thread/281754
[29]http://www.mathworks.com/help/techdoc/matlab_external/f24626.html#f29035
[30]http://www.mathworks.com/support/solutions/en/data/1-2RQL4L/
[31]https://code.ros.org/trac/opencv/browser/trunk/opencv/samples/c/find_obj.cpp?rev=2065

[32]http://seabee3-ros-pkg.googlecode.com/svn/trunk/cturtle/image_matcher/src/image_matcher.cpp
[33]https://code.ros.org/trac/opencv/browser/trunk/opencv/samples/cpp/generic_descriptor_match.cpp
[34] http://www.maths.lth.se/matematiklth/personal/petter/surfmex.php

[35]*More about SURF and feature detection*:
http://opencv.willowgarage.com/documentation/feature_detection.html
http://opencv.willowgarage.com/documentation/dynamic_structures.html
http://www.mathworks.com/matlabcentral/newsreader/view_thread/155296
http://www.mathworks.com/matlabcentral/newsreader/view_thread/152621
http://www.emgu.com/wiki/index.php/SURF_feature_detector_in_CSharp
https://code.ros.org/gf/project/opencv/scmsvn/?action=browse&path=%2Ftags%2F2.1%2Fopencv%2Fsamples%2Fc%2Ffind_obj.cpp&view=markup
http://opencv-users.1802565.n2.nabble.com/cvExtractSURF-td2126985.html